VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Řízení pohybu autocisteren na terminálu skladu pohonných hmot

Computer System for Automation and Control of Road Tanker Movement at Fuel Store Terminals

2010 Petr Havlena

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2010 .....

Rád bych na tomto místě poděkoval svému vedoucímu Ing. Davidu Bartuškovi a společnosti VAE CONTROLS, s.r.o. za možnost realizace této bakalářské práce.

Abstrakt

Tato bakalářská práce se zabývá vývojem řídícího systému pro společnost VAE CONTROLS, s.r.o. Cílem je analyzovat, navrhnout a implementovat systém pro simulaci a řízení pohybu vozidel na terminálu skladu pohonných hmot. Práce se věnuje důležitým aspektům pro popis reálného provozu skladů a problematice výdeje do autocisteren. Snahou je tento pracovní proces zdokonalit a ověřit, zda řešení bude přínosné. Realizace se zaměřuje na efektivní zpracování požadavků zákazníka a možnost testovat chování systému v nepředvídatelných situacích.

Klíčová slova: řídící systém, řízení, simulace, sklad pohonných hmot, plnění, autocisterna

Abstract

This bachelor thesis deals with a development of a control system for VAE CONTORLS, s.r.o. company. The main goal is to analyze, design and implement a system for automation and control of a vehicle movement at fuel store terminals. The work takes notice of all aspects important to a proper description of a real fuel store operation and fuel dispatching. The goal is to improve recent working procedures and verify the gains of the new system. Practical realization is concentrated on the effective processing of customer requests and testing of the system behavior in unpredictable situations.

Keywords: control system, control, automation, fuel store, fuel dispatching, road tanker

Seznam použitých zkratek a symbolů

PHM	–	Pohonné Hmoty a Maziva
UML	–	Unified Modeling Language
ERD	–	Entity Relationship Diagram
ORM	–	Objektově-Relační Mapování
OOP	–	Objektově Orientované Programování
LINQ	–	Language Integrated Query
SQL	–	Structured Query Language
T-SQL	–	Transact Structured Query Language
VB.NET	–	Visual Basic .NET

Obsah

1 Úvod **6**

2 Koncepce řídícího systému **7**
2.1 Poskytované funkce . . . 7
2.2 Architektura systému . . . 7
2.3 Výdej produktu . . . 7
2.4 Databáze . . . 8
2.5 Slovní zadání . . . 8

3 Specifikace požadavků **10**
3.1 Struktura skladu PHM . . . 10
3.2 Provoz . . . 10
3.3 Plánování . . . 10
3.4 Omezení provozu . . . 10
3.5 Prezentace výsledků . . . 10
3.6 Vstupy systému . . . 11
3.7 Výstupy systému . . . 11
3.8 Základní funkce . . . 11
3.9 Ostatní požadavky . . . 12
3.10 Případy užití . . . 12

4 Analýza systému **14**
4.1 Konceptuální analýza . . . 14
4.2 Datová analýza . . . 15
4.3 Funkční a dynamická analýza . . . 15

5 Návrh implementace **22**
5.1 Přístup k datům . . . 22
5.2 Paralelní zpracování . . . 23

6 Popis implementace **25**
6.1 Použité technologie . . . 25
6.2 Datová vrstva . . . 25
6.3 Synchronizace vláken . . . 27
6.4 Logování . . . 28

7 Testování aplikace **29**
7.1 Unit testy . . . 29

8 Instalace a provoz systému **30**
8.1 Požadavky na programové vybavení . . . 30
8.2 Průběh instalace . . . 30
8.3 Provoz aplikace . . . 30

9 Zhodnocení výsledků **31**

10 Závěr **33**

11 Literatura **34**

Přílohy **34**

A Obsah přiloženého CD **35**

B Uživatelská příručka **36**
B.1 Instalace 36
B.2 Konfigurační soubor 36
B.3 Kontrolní výpisy 36

C Programátorská příručka **37**
C.1 Adresářová struktura 37
C.2 Konfigurace služby pro logování 37

D Datový slovník **38**

E Grafy průběhu plnění **40**

F Zdrojové kódy **44**
F.1 Třída Logger 44
F.2 Třída Repository 45
F.3 Třída BlockingPriorityQueue 47
F.4 Třída Worker 48

Seznam tabulek

1 Část datového slovníku 15
2 Konfigurace stop 32
3 Kategorie vozidel 32
4 Tabulka FuelStore 38
5 Tabulka LoadingBay 38
6 Tabulka ProductOnLoadingBay 38
7 Tabulka Product 38
8 Tabulka ProductClass 38
9 Tabulka Vehicle 39
10 Tabulka VehicleChamber 39

Seznam obrázků

1 Architektura systému 9
2 Případ užití – Základní funkce 12
3 Případ užití – Zařazení požadavku do fronty 13
4 Případ užití – Archivace výsledků 13
5 Třídní diagram 16
6 Diagram aktivit – Plánování trasy 18
7 Diagram aktivit – Generování požadavku 19
8 Sekvenční diagram – Výdej do autocisterny 20
9 Stavový diagram – Rozpis plnění 21
10 Repository třída 24
11 Fronta producent/spotřebitel 24
12 Časy plnění autocisteren o objemu 2900 l 40
13 Časy plnění autocisteren o objemu 12 500 l 41
14 Časy plnění autocisteren o objemu 32 000 l 42
15 Časy plnění autocisteren o objemu 40 777 l 43

Seznam výpisů zdrojového kódu

1 Ukázka jazyka LINQ 25
2 Rozhraní IRepository 26
3 Třída BlockingQueue 27
4 Příklad použití knihovny log4net 28
5 Konfigurace knihovny log4net 37
6 Třída Logger 44
7 Třída Repository 45
8 Třída BlockingPriorityQueue 47
9 Třída Worker 48

1 Úvod

Ropa je strategickou energetickou a chemickou surovinou. Vyskytuje se ve všech odvětvích průmyslové výroby a je základním palivem pro dopravu. Zajištění dostatečných rezerv je nezbytným úkolem pro zachování stability.

Sklady pohonných hmot jsou zařízení určená k uchovávání a distribuci ropných produktů. Areál skladů tvoří nadzemní a podzemní zásobníky, manipulační nádrže, plnící lávky automobilových cisteren, objekty pro stáčení a plnění železničních vagónů, technologické rozvody, laboratoře, požární nádrže, elektrické rozvodny aj. Zboží se dále ze skladu přepravuje autocisternami, železnicí nebo produktovodem.

Střediska se často nacházejí v blízkosti měst a obcí, proto je prioritou zajištění maximální bezpečnosti. Každý stát musí pro případy krizových situací (živelních pohrom a jiných, například v době ropné krize) držet nouzové zásoby ropy a ropných produktů a musí být schopen tyto zásoby na svém území efektivně a spolehlivě distribuovat. Skladové kapacity toto umožňují realizovat.

Na podnět společnosti VAE CONTROLS, s.r.o., která projektuje a realizuje komplexní dodávky technologií, řídicích systémů a elektronických zařízení pro sklady pohonných hmot, vznikla tato práce. Popisuje všechny fáze vývoje systému pro řízení a automatizaci činností probíhajících na skladovém terminálu. Jednotlivé části korespondují s posloupností kroků softwarového procesu. Nejprve je provedena specifikace požadavků, následuje analýza, návrh a implementace. Závěrečné kapitoly shrnují zkušenosti s provozem systému a jeho přínos.

2 Koncepce řídícího systému

Sklady PHM jsou technologické celky, které z hlediska svého zaměření kladou vysoké nároky na spolehlivost a bezpečnost řídících systémů a na rychlý přístup k technologickým datům. Práce s vysoce hořlavými a nebezpečnými látkami požaduje zabezpečit spolehlivý chod i při výjimečných situacích jako jsou výpadky napájení nebo selhání některých komponent systému.

2.1 Poskytované funkce

Firma VAE CONTROLS, s.r.o. v současné době vyvíjí a dodává komplexní systém pro řízení terminálů a skladů pohonných hmot. Hlavní funkce řídícího systému jsou:

- automatizace technologického procesu,
- řešení mimořádných situací,
- vizualizace a dispečerské řízení technologie,
- tisk dodacích listů, protokolů,
- evidence zákazníků a jejich objednávek,
- evidence řidičů, vozidel, transakcí atd.,
- komunikace s jinými informačními systémy jako jsou ekonomický, manažerský apod.,
- archivace veškerých událostí.

2.2 Architektura systému

Z důvodu zajištění spolehlivosti je systém realizován pomocí třístupňové architektury. Nejnižší stupeň zahrnuje veškerá technologická zařízení skladu, která mají být připojena. Další stupeň představují programovatelné automaty (PLC), na něž jsou přivedeny všechny vstupy, výstupy a komunikační linky z technologických zařízení. Nejvyšší stupeň tvoří pracovní stanice s architekturou klient/server, které poskytují uživatelské rozhraní pro řízení a sledování technologie a provádění transakcí se zbožím.

2.3 Výdej produktu

Proces výdeje produktu z výdejních lávek je automatizován. Systém zajišťuje identifikaci zákazníka a poskytuje aktuální informace o průběhu plnění a stavu výdejových tras. Samotný pohyb autocisteren na terminálu řízení nepodléhá. Analýzu požadavku zákazníka a přidělení ideální trasy pro odběr produktu řešení nezahrnuje. Rostoucí nároky na kapacitu a plynulost provozu skladů PHM dávají podnět tuto funkcionalitu implementovat.

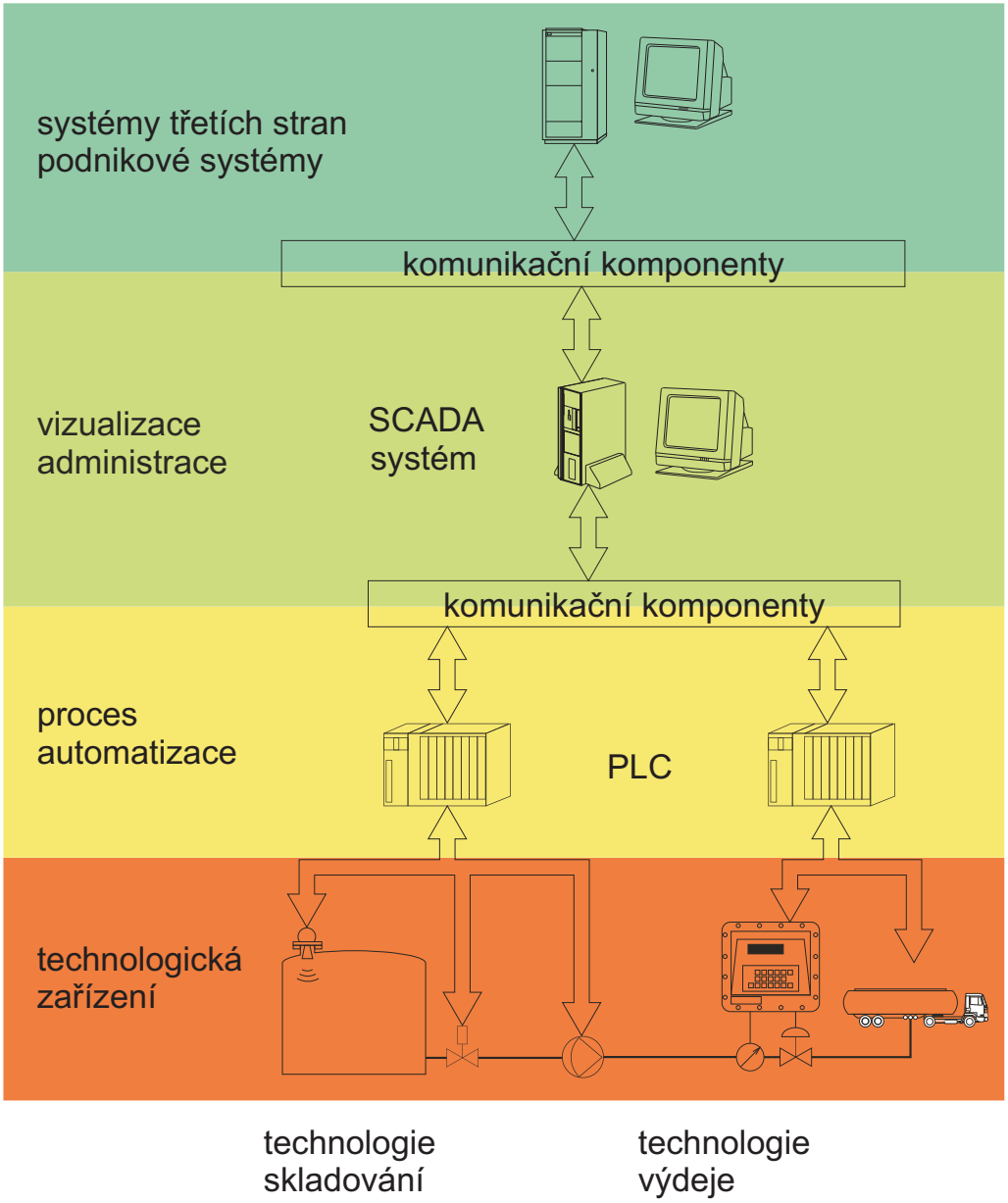
2.4 Databáze

Do databáze jsou pomocí aplikační logiky ukládány veličiny týkající se transakcí, které vzniknou při pohybu zboží na skladě (např. výdej do autocisterny, příjem z železnice). Tyto transakce mohou být dále zpracovány nebo exportovány do jiného informačního systému třetích stran. Dále zde běží úlohy, které sledují pohyb řidičů na skladě podle informací předaných z identifikačního systému.

Další součástí je evidence řidičů, dopravců, vlastníku, míst určení a vozidel týkajících se transferu pohonných hmot, výdejových a příjmových tras a atestů. Tyto a jiné údaje lze doplňovat a upravovat. Strukturovaná data se dají dobře využít jako prostředek k popisu reality. Je možné vyčlenit důležité parametry a definovat, jakým způsobem se navzájem ovlivňují.

2.5 Slovní zadání

Pokuste se o zdokonalení procesu plnění autocisteren externím systémem. Aplikace bude představovat funkční model skladu s odpovídajícími vstupy a výstupy. Pohyb na terminálu založen na vůli řidičů má být nahrazen prostředky, které generují scénář podle konkrétní situace. Zaměřte se na chování při výskytu chyby s ohledem na co největší plynulost provozu. Navrhněte uživatelské rozhraní s možností modifikace vstupních parametrů. Výsledky vhodně prezentujte a zvažte možnost vizualizace dat. Předpokládá se řešení v souladu s principy stávajícího řídícího systému.



Obrázek 1: Architektura systému

3 Specifikace požadavků

Úkolem specifikace je popis nároků na systém pomocí terminologie, které rozumí uživatel i programátor. Požadavky zadavatele je vhodné definovat ve formě strukturovaného textu, doplněného o potřebné vstupy, výstupy a funkce.

3.1 Struktura skladu PHM

Cílem je navrhnout aplikaci pro simulaci a řízení procesu výdeje do autocisteren. Model skladu musí uvažovat všechny skutečnosti nezbytné pro správné vyhodnocení situace a odbavení zákazníka. K realizaci plnění je k dispozici několik stop. Každá stopa vyřizuje současně požadavek právě jedné autocisterny a obsahuje konfiguraci pro výdej jednoho nebo více produktů. Čekající vozidla se řadí do fronty na terminálu u konkrétní stopy, pokud to kapacita dovoluje, nebo zůstávají u vjezdu do skladu.

3.2 Provoz

V průběhu dne dojde k vyřízení přibližně 100 požadavků na odběr produktu. Zákazníci přijíždějí v nepravidelných intervalech, je pravděpodobnost na dočasně zvýšenou hustotu provozu. Autocisterna zpravidla obsahuje několik komor, které určují její celkový objem. Typicky se objem cisterny pohybuje v rozmezí 10 000–42 000 litrů. Celková doba plnění je dána poměrem požadovaného množství a maximálního průtoku stopy, dále časem potřebným k uzemnění vozidla.

3.3 Plánování

Algoritmus má po příjezdu zákazníka rozhodnout o trase, pořadí a způsobu naplnění jednotlivých komor na základě aktuálních provozních vlastností skladu. Cílem je efektivně zpracovávat frontu požadavků a zajistit rovnoměrné zatížení terminálu. Prodlevy způsobené nevhodným využitím zdrojů jsou nežádoucí. Přidělení stopy bude probíhat na základě dostupných produktů, využité kapacity a výskytu poruchy.

3.4 Omezení provozu

V průběhu provozu může nastat situace, že některá zařízení nebudou pracovat správně. V takovém případě je nutné tuto skutečnost zahrnout do procesu plánování a podle potřeby jej přehodnotit. Porucha stopy způsobí, že vozidla k ní přiřazená se musí přesunout. Porucha produktu má za následek jeho nedostupnost na všech stopách. Pokud se objeví okolnosti, které způsobí vyřazení celého skladu, veškerý provoz se zastaví až do okamžiku, kdy je funkce opět obnovena.

3.5 Prezentace výsledků

Aplikace musí poskytovat výsledky ve formě vhodné pro další zpracování. Analýza dat a srovnání s údaji, které odráží současný stav, určí přínos výsledného řešení. Pro získání

dostatečného vzorku je vhodné upravit poměr rychlosti generování požadavků a reálného času. Zároveň však data musí zůstat dobře čitelná.

3.6 Vstupy systému

Seznam skladů: název skladu, adresa, omezení maximálního průtoku.

Informace o stopách: maximální průtok, kapacita fronty vozidel.

Dostupné produkty: název, místní název, zkratka, alternativní název, průměrná hustota.

Třída produktu: název, popis.

Seznam vozidel: registrační značka, typ, výrobce, model, počet komor, počet náprav, užitečná hmotnost, celková hmotnost, pohotovostní hmotnost, celkový objem, umístění na terminálu, čas posledního plnění.

Komory vozidla: pořadové číslo, objem.

3.7 Výstupy systému

Rozpis plnění: obsahuje údaje popisující proces výdeje (čas příjezdu, čas odjezdu, aktuální stav).

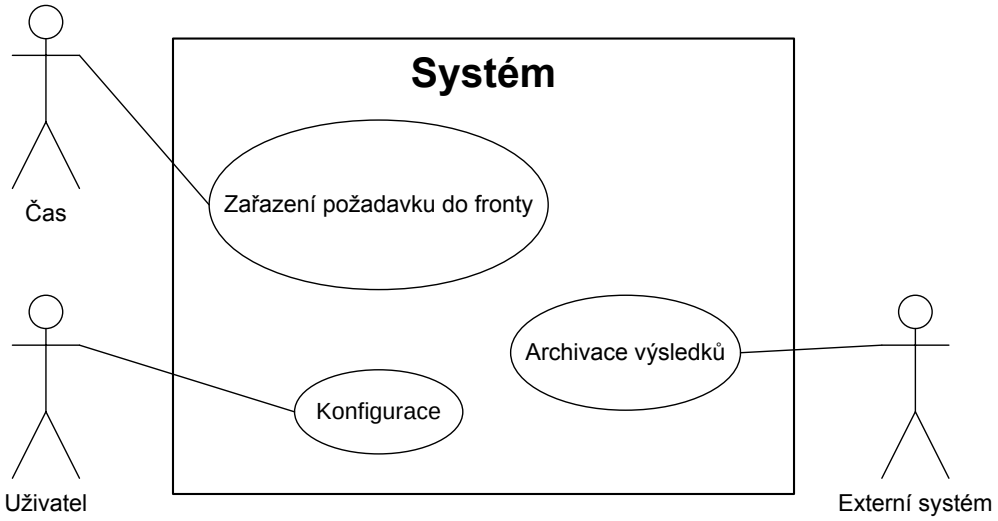
Komory rozpisu plnění: průběh zpracování každé komory (požadované množství, odebrané množství, aktuální stav, pořadí plnění).

Logovací záznamy: informace o situaci na jednotlivých stopách, zatížení skladu a výskytu poruchy.

3.8 Základní funkce

Očekává se, že budou k dispozici následující funkce:

- vytvoření modelu skladu na základě uživatelské konfigurace,
- simulace příjezdu vozidla,
- generování rozpisu plnění,
- vyhodnocení požadavku a plánování trasy,
- přidání požadavku do fronty,
- simulace výdeje do autocisterny (časové prodlevy musí odpovídat realitě),
- generování poruchy,
- logování událostí,
- archivace výsledků.



Obrázek 2: Případ užití – Základní funkce

3.9 Ostatní požadavky

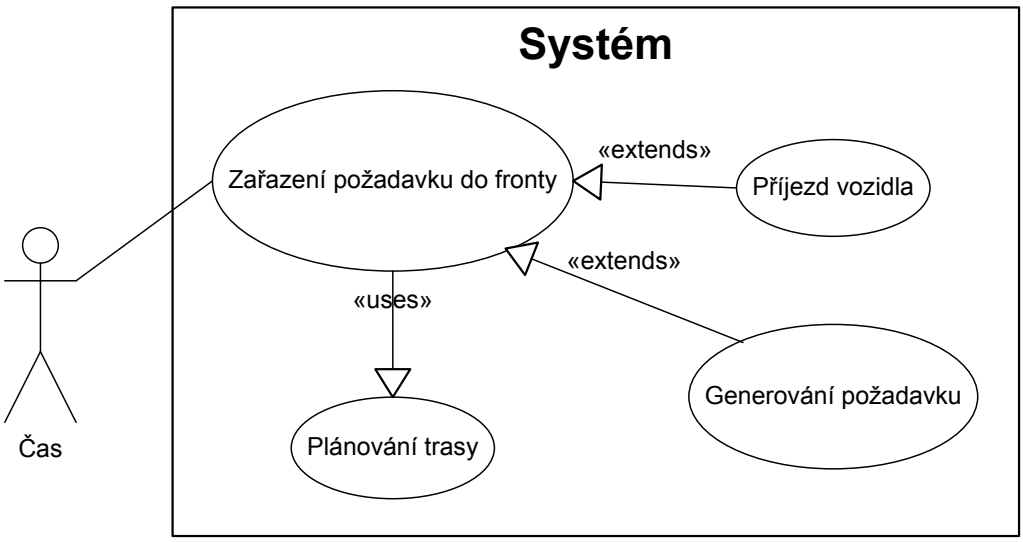
V první fázi bude systém sloužit interním potřebám firmy pro analýzu provozu skladů PHM. Výstupy budou zákazníkům prezentovány jiným způsobem. Je vhodné, aby aplikace poskytovala struktury vhodné pro pozdější možnost vizualizace. Uživatelská nastavení budou probíhat pomocí konfiguračních souborů, výsledky budou ukládány do databáze. Řešení bude koncipováno jako služba pro účely dlouhodobější simulace. Uživatel získá informace o činnosti aplikace pomocí nástroje pro logování.

3.10 Případy užití

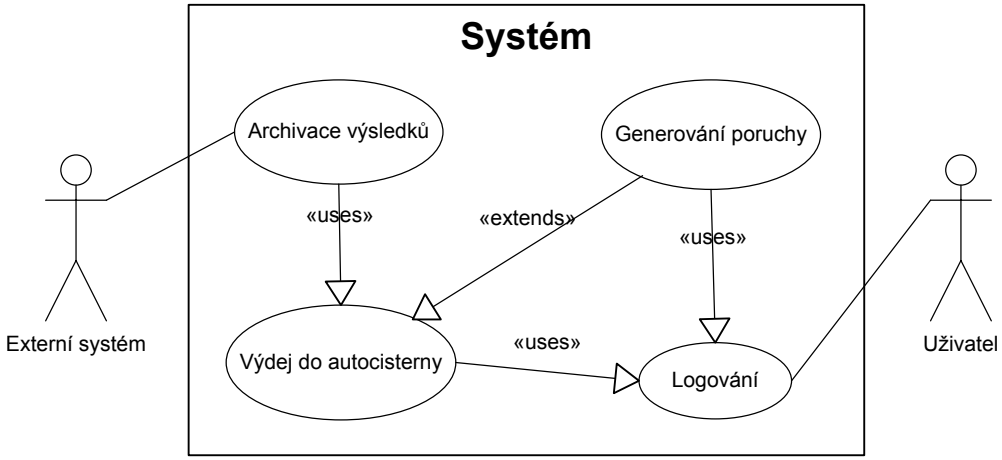
Modelování případů užití je další formou inženýrství požadavků; tuto aktivitu lze rozdělit na následující etapy:

- nalezení hranic systému,
- nalezení účastníků,
- nalezení případů užití.

Modely případů užití poskytují hlavní zdroj objektů a tříd. Jsou prvotním vstupem k modelování tříd.



Obrázek 3: Případ užití – Zařazení požadavku do fronty



Obrázek 4: Případ užití – Archivace výsledků

4 Analýza systému

Analýza představuje další fázi procesu vývoje softwaru a pokrývá převod slovního zadání či jinak zpřesněného popisu reality do tvaru, který může sloužit jako podklad pro zahájení návrhu řešení a pro vlastní implementaci. Záměrem analýzy je tvorba analytického modelu, jenž zachycuje podstatné požadavky a charakteristické rysy systému. Tento model se zaměřuje na to, co systém musí udělat, avšak nezabývá se detaily týkajícími se způsobu, jakým to udělá. Analýza se skládá z několika částí, kde každá se zaměřuje na určité aspekty chování systému. Mezi tyto části patří:

- konceptuální analýza,
- datová analýza,
- funkční a dynamická analýza.

4.1 Konceptuální analýza

Konceptuální modelování je proces vývoje sémantického popisu systému, který je uplatněn při návrhu a implementaci databázové aplikace. Je využito prostředků jazyka UML, který oproti ER modelu přináší více objektově orientovaný přístup k návrhu databáze. Poskytuje stejný jazyk pro analýzu databáze i samotné aplikace a je použitelný pro objektově-relační i relační datový model.

4.1.1 Třídní diagram

Diagram tříd poskytuje statický pohled na strukturu systému. Modeluje entity jako třídy s atributy a chováním. Definuje vazby a omezení, které existují mezi třídami.

4.1.2 Lineární zápis typů entit

Primární klíče jsou podtrženy, u cizích klíčů je provedena sazba italikou.

FuelStore (fuelStoreId, fuelStoreName, address1, address2, address3, maxLitersPerMinute, cancelled)

LoadingBay (loadingBayId, maxLitersPerMinute, *fuelStoreId*)

ProductOnLoadingBay (*loadingBayId*, *productId*)

Product (productId, productName, localName, shortName, altName, avgDensity, *productClassId*)

ProductClass (productClassId, description)

Vehicle (vehicleId, regNumber, vehicleType, manufacturer, model, numChambers, numAxles, limitWeight, totalWeight, emptyWeight, maxVolume, currentState, lastLoading)

Atribut	Datový typ	Klíč	NULL	Index	Popis
Tabulka LoadingSchedule					
scheduleId	int	A	N	A	číslo rozpisu plnění
arrivalTime	datetime	N	A	N	čas příjezdu
departureTime	datetime	N	A	N	čas odjezdu
currentState	nvarchar(15)	N	A	N	aktuální stav
fuelStoreId	int	N	N	A	reference na sklad
vehicleId	int	N	N	A	reference na vozidlo
Tabulka LoadingScheduleChamber					
chamberId	int	A	N	A	číslo komory
requestedVolume	decimal(12, 3)	N	A	N	požadované množství
loadedVolume	decimal(12 ,3)	N	A	N	odebrané množství
currentState	nvarchar(15)	N	A	N	aktuální stav
loadingInProgress	bit	N	A	N	příznak průběhu plnění
loadingOrder	int	N	A	N	pořadí plnění
scheduleId	int	A	N	A	cizí klíč – tab. LoadingSchedule
productId	int	N	N	A	reference na produkt
loadingBayId	int	N	N	A	reference na stopu

Tabulka 1: Část datového slovníku

VehicleChamber (chamberId, volume, *vehicleId*)

LoadingSchedule (scheduleId, arrivalTime, departureTime, currentState, fuelStoreId, vehicleId)

LoadingScheduleChamber (chamberId, requestedVolume, loadedVolume, currentState, loadingInProgress, loadingOrder, *scheduleId*, productId, loadingBayId)

4.2 Datová analýza

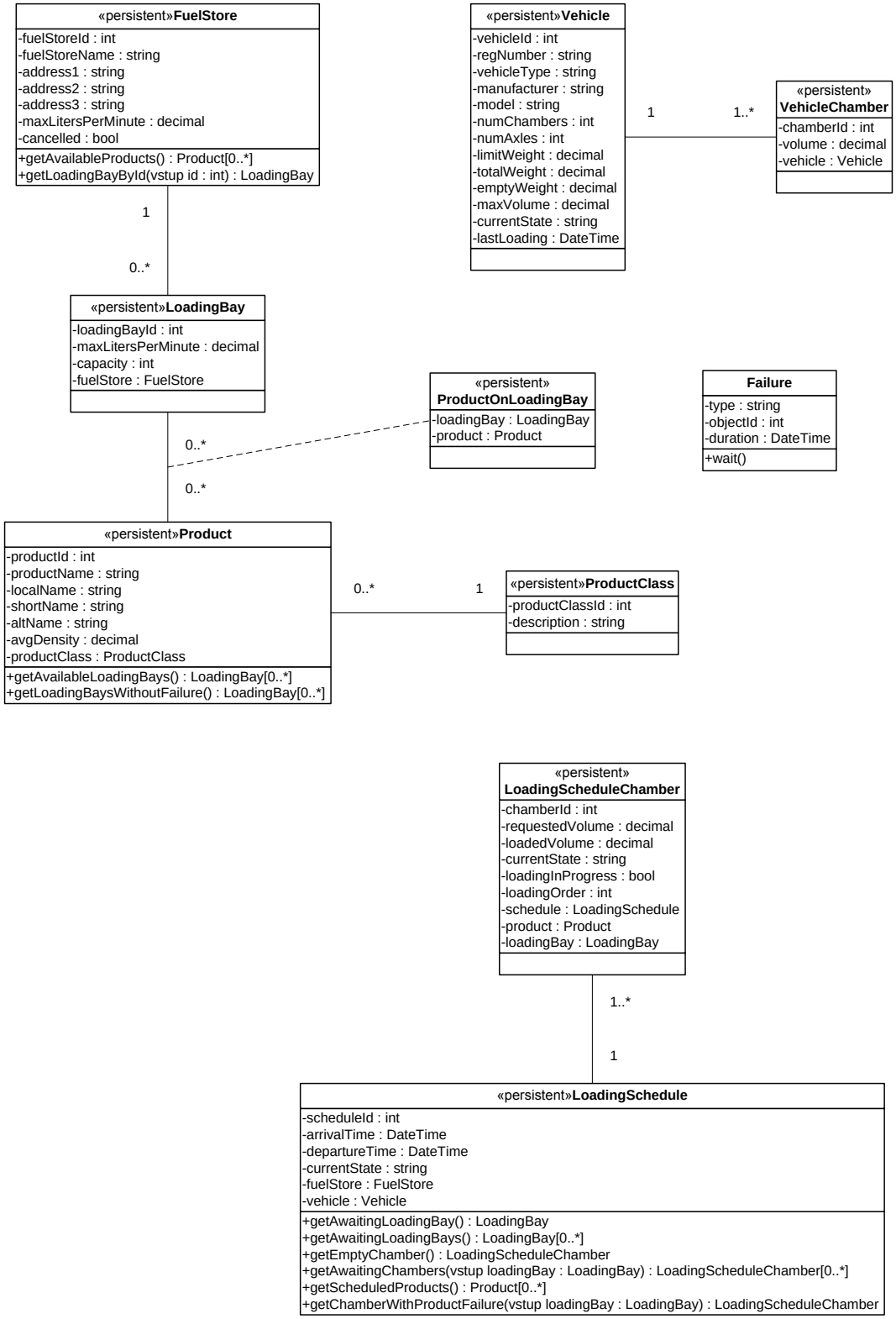
Mezi důležité části návrhu systému, jehož součástí je databáze, patří volba vhodného uložení dat, popis tabulek a jejich atributů včetně definice vztahů mezi nimi.

4.2.1 Datový slovník

Datový slovník obsahuje podrobnou specifikaci atributů každé databázové tabulky. Jsou zde uvedeny datové typy, integritní omezení a další informace. V tabulce 1 je uvedena část datového slovníku, která popisuje tabulky *LoadingSchedule* a *LoadingScheduleChamber*.

4.3 Funkční a dynamická analýza

Funkční a dynamická analýza popisuje všechny procesy probíhající v systému včetně jejich vzájemných závislostí a chování v průběhu času. Pro tvorbu funkčního modelu jsou použity především tyto nástroje:



Obrázek 5: Třídní diagram

- diagram aktivit,
- sekvenční diagram,
- stavový diagram.

4.3.1 Diagramy aktivit

Diagramy aktivit jsou grafy toků ukazující úlohy, které je nutné vykonat ve výpočetním procesu. Lze je s úspěchem použít rovněž k modelování obchodních (podnikatelských) pracovních postupů.

4.3.1.1 Proces plánování trasy Určení místa a pořadí výdeje je jedním ze základních úkolu řídícího systému. Základní popis algoritmu ukazuje obrázek 6.

4.3.1.2 Generování požadavku Generování rozpisů plnění a jejich zpracování probíhá paralelně. Obě úlohy se neustále opakují, dokud nepřijde podnět k jejich ukončení (obrázek 7).

4.3.2 Sekvenční diagram

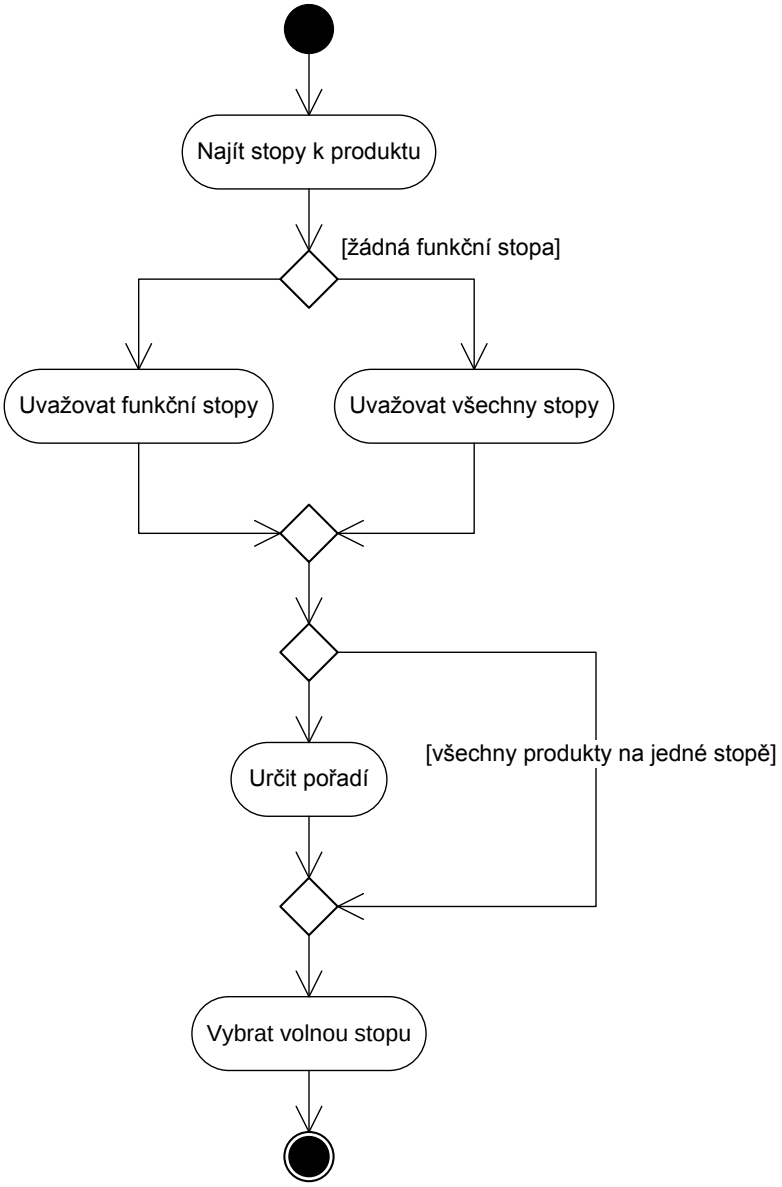
Sekvenční diagramy slouží k popisu interakcí ve vztahu k času. Jejich výhodou je chronologické zobrazení předaných zpráv jako časově uspořádané posloupnosti událostí.

4.3.2.1 Výdej do autocisterny Průběh výdeje je třeba dobře zmapovat, neboť vyžaduje provádět činnosti ve specifickém pořadí. Fronta požadavků je hlavním prostředkem pro komunikaci mezi instancemi.

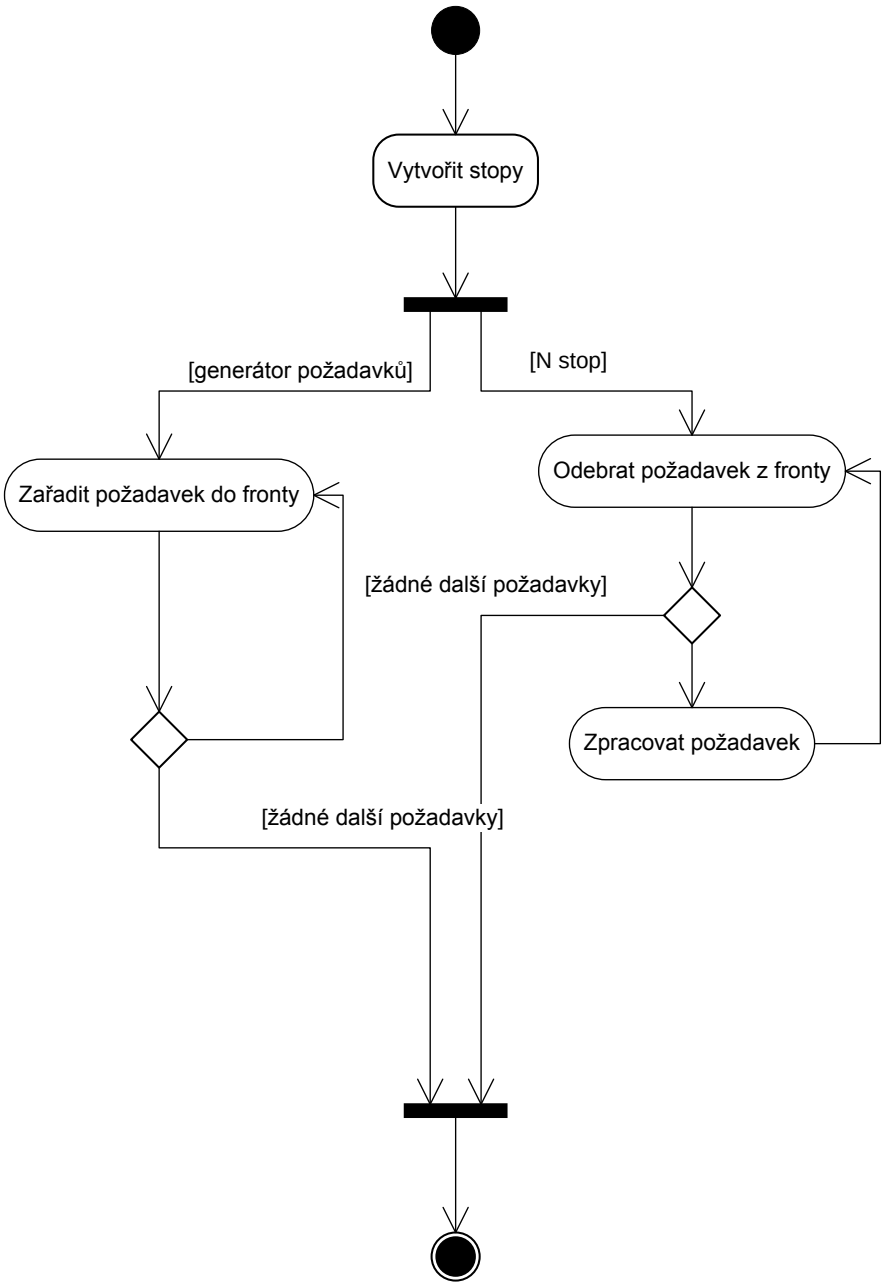
4.3.3 Stavový diagram

Stavové diagramy jsou důležitou pomůckou při modelování dynamického chování. Používají se k zobrazení životního cyklu jednoho objektu.

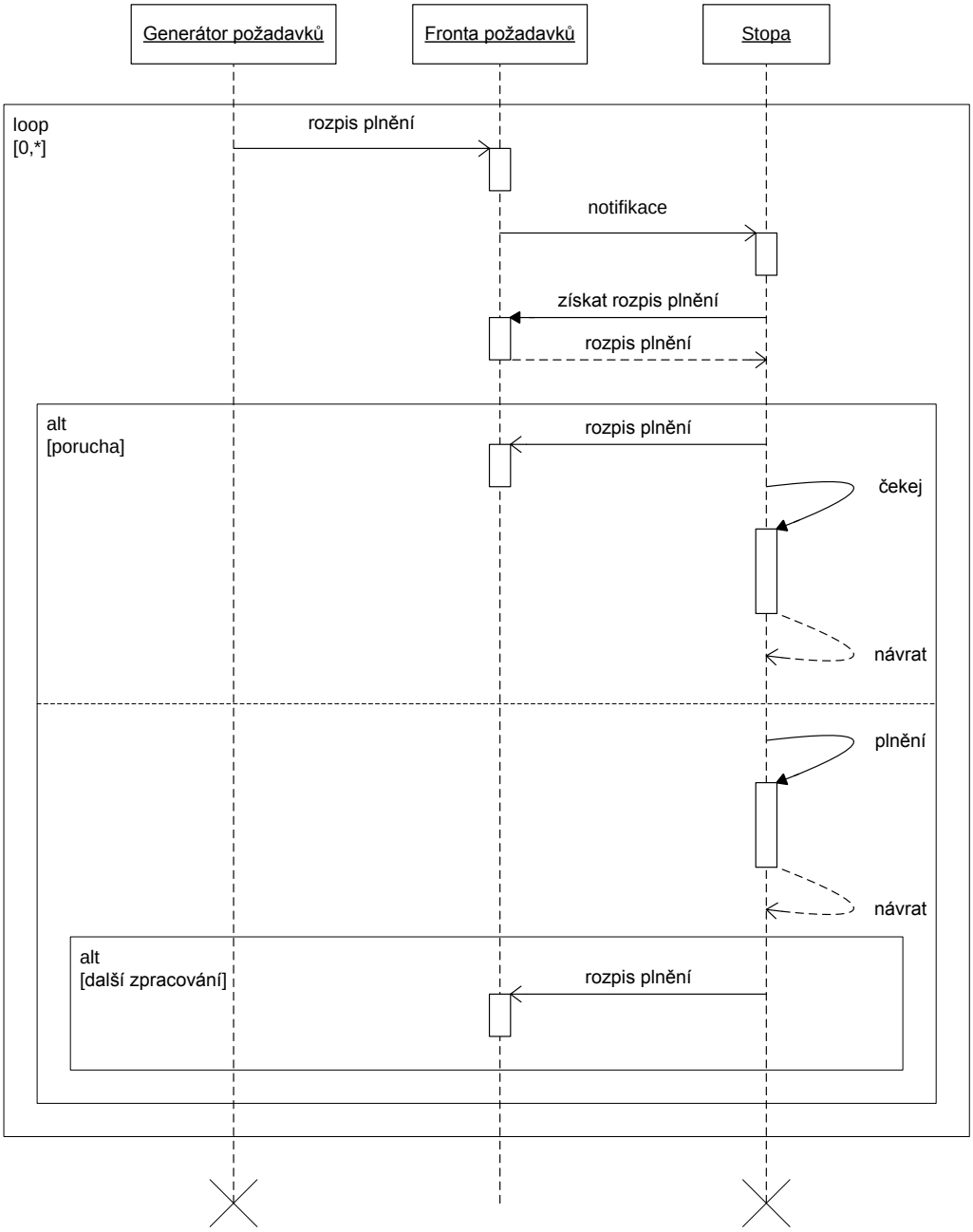
4.3.3.1 Rozpis plnění Rozpis plnění představuje úlohu, jejíž stav se v průběhu zpracování mění. V každém okamžiku poskytuje informaci o právě probíhající fázi výdeje.



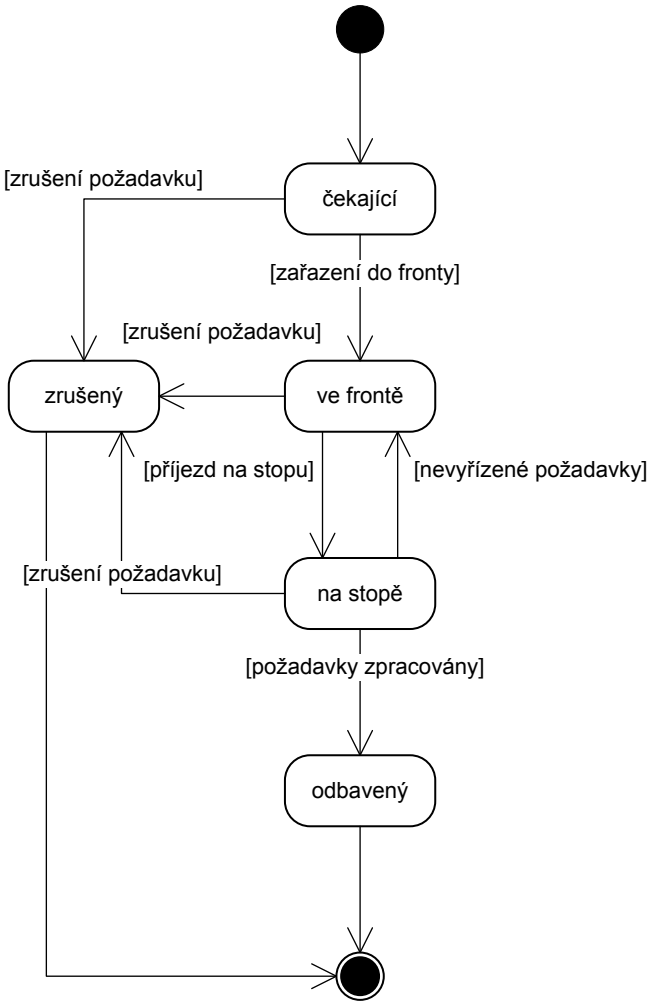
Obrázek 6: Diagram aktivit – Plánování trasy



Obrázek 7: Diagram aktivit – Generování požadavku



Obrázek 8: Sekvenční diagram – Výdej do autocisterny



Obrázek 9: Stavový diagram – Rozpis plnění

5 Návrh implementace

Analýza je zaměřena hlavně na tvorbu logického modelu připravovaného systému, který zachycuje funkce, jež tento systém musí poskytovat, aby uspokojil požadavky uživatelů. Smyslem návrhu je přesná specifikace způsobu, jak takové funkce implementovat. Na tuto otázku se lze dívat z pohledu problémové domény, ale i z pohledu domény řešení. Požadavky přicházejí z problémové domény. Analýza je vlastně zkoumáním této domény z pohledu uživatelů systému a dalších zainteresovaných osob. Návrh spočívá ve sloučení technických řešení z domény řešení (knihovny tříd, mechanismy perzistence apod.) za účelem vytvoření modelu systému (návrhového modelu), který skutečně lze implementovat.

Během návrhu rozhodují návrháři objektově orientovaného systému o strategických otázkách, např. o perzistenci objektů, o distribuci a o tvorbě příslušného návrhového modelu.

5.1 Přístup k datům

Pro každý systém obsahující databázi je třeba navrhnout sofistikovaný způsob získávání a ukládání dat. Tyto postupy umožňují aplikaci rozdělit do více vrstev, což je potřebné zejména u větších projektů kvůli jejich celkové přehlednosti.

5.1.1 Perzistence objektů

Objekty se dají označit jako perzistentní, pokud je možné je uložit do souboru, databáze, nebo obecně kamkoliv mimo operační paměť, a později je opět načíst beze ztráty jejich vlastností či vztahů mezi nimi. Tato vlastnost bude zajištěna nástrojem pro objektově-relační mapování.

5.1.2 Objektově-relační mapování

Objektově-relační mapování slouží k tomu, aby bylo možné snadno používat relační databáze v prostředí objektově orientovaných programovacích jazyků. Vzhledem k tomu, že objektově orientovaný návrh dat není jednoznačně převoditelný na relační databáze a opačně, používají se různé formy mapování.

Mapování má za účel načítat data z relační databáze a naplnit jimi příslušné datové položky objektů, případně naopak datové položky objektů ukládat do databáze. Snahou ORM je co nejlepší využití obou zmíněných technologií – tj. objekty by měly reprezentovat prvky reálného světa, jak to požadují principy OOP, na straně databáze bychom zase měli využít všech možností relačních databází – indexy, pohledy, primární klíče, atd.

5.1.3 Repository třídy

Repository třídy se někdy označují také jako DAO (Data Access Object). Oba tyto výrazy se používají pro třídy, které se starají o ukládání, načítání a další operace související

s perzistentními třídami. Model aplikace neví, jaká třída se stará o práci s daty. Díky tomu, že persistenci zajišťuje cizí objekt, stačí nám znát pouze jeho rozhraní a v případě potřeby ho lze snadno nahradit jiným.

Obvyklá praxe je pro každou perzistentní třídu vytvořit příslušnou repository třídu. Všechny repository třídy dohromady vytvářejí repository vrstvu aplikace, čili vrstvu, která se soustředí na perzistenci objektů.

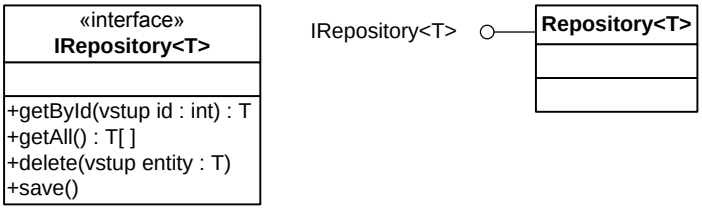
5.2 Paralelní zpracování

Vícevláknové aplikace poskytují základ pro efektivní zpracování časově náročných úloh. Použití vláken však s sebou přináší nutnost řešit jejich synchronizaci. Špatné pochopení problematiky může vést k nestabilnímu programu. Z tohoto důvodu je vhodné držet se doporučených postupů.

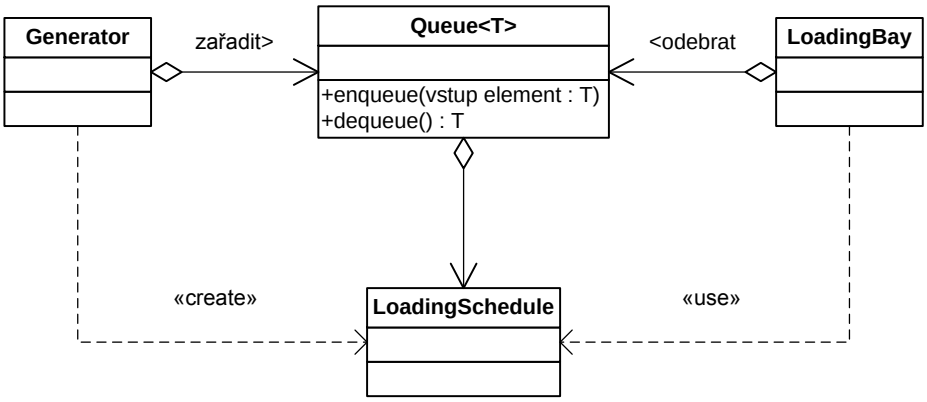
5.2.1 Fronta producent/spotřebitel

Návrhový vzor producent/spotřebitel najde uplatnění v situaci, kdy je třeba požadavky zpracovávat ve specifickém pořadí. Producent generuje úkoly a spotřebitelem je proces, který tyto úkoly plní. Frontu požadavků může obsluhovat několik spotřebitelů, to je výhodné pro paralelní zpracování a využití potenciálu víceprocesorových systémů.

V našem případě bude producent simulovat příjezd vozidla a generovat rozpis plnění, který se zařadí do fronty a provede se notifikace jednotlivých stop – spotřebitelů, zajišťujících výdej. Realizuje se plnění a požadavek je odbaven. Pokud je fronta prázdná, stopy čekají na signál, který indikuje přidání nového úkolu.



Obrázek 10: Repository třída



Obrázek 11: Fronta producent/spotřebitel

6 Popis implementace

Implementace spočívá v převodu návrhového modelu do spustitelného kódu. Výsledkem této etapy by měl být funkční systém, který bude splňovat požadavky uvedené v počátku softwarového procesu. Před vlastní implementací je ještě nutné specifikovat vývojové a aplikační prostředí.

6.1 Použité technologie

Výběr programových prostředků probíhal na základě požadavků objektově orientovaného návrhu, oddělení datové vrstvy a aplikační logiky a moderního rozhraní pro vývoj aplikací.

Platforma: .NET Framework 3.5

Programovací jazyk: C# 3.0

Databázový server: SQL Server 2008

6.2 Datová vrstva

Datová vrstva obsahuje funkce pro přístup k informacím v datovém úložišti. Její správný návrh poskytuje nezávislost aplikace na zdroji dat.

6.2.1 LINQ to SQL

LINQ je integrovaný jazyk pro dotazování, který byl představen spolu s jazyky C# 3.0 a Visual Basic 9 spolu s .NET Frameworkem 3.5. Umožňuje manipulovat s jakýmikoliv daty pomocí nových klíčových slov a podpůrných jazykových konstrukcí.

LINQ to SQL je nástroj pro objektově-relační mapování, který umožňuje používat konstrukce jazyka LINQ. Podporuje pouze databázový systém SQL Server. V tradičních ORM nástrojích je konceptuální model mapován na fyzickou strukturu. LINQ to SQL vždy mapuje jednu tabulku na jeden objekt. Technologie umožňuje vyhnout se používání T-SQL v datové vrstvě, kód v C#/VB.NET bude automaticky převeden na příslušné SQL dotazy.

LINQ to SQL lze používat dvěma způsoby. Prvním je vytvořit si definice tříd a atributů a dynamicky vygenerovat databázi. Druhý způsob představuje konverzi tabulek, pohledů a uložených procedur do tříd kódu, kde veřejné vlastnosti odpovídají jednotlivým sloupcům tabulek. Objekty dané třídy pak reprezentují řádky tabulky.

```
int [] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };

var lowNums =
    from n in numbers
    where n < 5
    select n;
```

```
Console.WriteLine("Numbers < 5:");
foreach (var x in lowNums)
{
    Console.WriteLine(x);
}
```

Výpis 1: Ukázka jazyka LINQ

6.2.2 Třída DataContext

`DataContext` třída zajišťuje komunikaci s databází. Je to zdroj všech objektů dostupných pomocí databázového připojení. Existují různé strategie vytváření instance `DataContext` objektu:

- jedna instance pro celou aplikaci,
- jedna instance pro každé vlákno,
- nová instance pro každý datový objekt,
- nová instance při každém provedení dotazu.

Aplikace uchovává `DataContext` v rámci Repository tříd, které reprezentují logickou množinu souvisejících databázových operací.

6.2.3 Databázové operace

Získávání, vkládání, upravování a mazání dat probíhá pomocí tříd, které implementují rozhraní `IRepository<T>`. Toto rozhraní poskytuje určitou míru abstrakce mezi doménovým modelem a datovou vrstvou. Pro scénáře, kdy kód zůstává stejný, ale mění se typy, je vhodné použití generických datových typů. Generika poskytují zjednodušení často se opakujících úkolů a přidávají podporu typové kontroly. Vzorová implementace rozhraní `IRepository<T>` se nachází v příloze F.2.

```
public interface IRepository<T> where T : class
{
    T GetById(int id);
    IQueryable<T> GetAll();
    void InsertOnSubmit(T entity);
    void DeleteOnSubmit(T entity);
    void DeleteAllOnSubmit(IEnumerable<T> entities);
    void SubmitChanges();
}
```

Výpis 2: Rozhraní IRepository

6.3 Synchronizace vláken

Zamykání vláken je základním prostředek pro zajištění vláknové bezpečnosti. Zamykání zajišťuje exkluzivní přístup k dané části kódu, tedy, že v jednu chvíli může k vymezenému zdroji přistupovat jen jedno vlákno. Tento postup zabraňuje nepředvídatelnému chování aplikace.

Interakce mezi vlákny probíhá pomocí sdílené fronty požadavků. Většina referenčních typů, které poskytuje .NET Framework, nezaručuje vláknovou bezpečnost. Toto omezení existuje z důvodu výkonu. Riziková místa je třeba ošetřit explicitně.

6.3.1 Vláknově bezpečné kolekce

Jak již bylo řečeno, metody, které vestavěné kolekce poskytují, nejsou reentrantní. Zároveň potřebujeme mechanismus pro posílání signálů pracovnímu vláknu v případě, že je fronta požadavků prázdná nebo jako podnět k činnosti. Tyto vlastnosti implementuje třída `BlockingQueue<T>`. Aplikace používá její odvození, které umožňuje přidání prvku s určitou prioritou. Kompletní zdrojový kód poskytuje příloha F.3.

6.3.2 Synchronizační konstrukce

Jednu z možných signalizační konstrukce poskytuje třída `Monitor` pomocí statických metod `Wait` a `Pulse`. `Wait` zablokuje vlákno, dokud nedostane z jiného vlákna signál od metody `Pulse`. Před použitím `Wait/Pulse` je třeba definovat synchronizační objekt. Platí, že pokud obě vlákna používají stejný synchronizační objekt, můžou si mezi sebou posílat signály. Druhou důležitou věcí je, že synchronizační objekt musí být vždy uzamknut, než ho použijeme při volání `Wait` nebo `Pulse`. Šablonu pro zapouzdření některých funkcí poskytovaných vlákny zahrnuje příloha F.4.

```
public class BlockingQueue<T>
{
    private Queue<T> queue = new Queue<T>();

    public void Enqueue(T element)
    {
        lock (queue)
        {
            queue.Enqueue(element);
            Monitor.PulseAll(queue);
        }
    }

    public T Dequeue()
    {
        lock (queue)
        {
            while (queue.Count == 0)
            {
                Monitor.Wait(queue);
            }
```

```
            return queue.Dequeue();
        }
    }

    public int Count()
    {
        lock (queue) return queue.Count;
    }
}
```

Výpis 3: Třída BlockingQueue

6.4 Logování

Každá aplikace by měla poskytovat mechanismus pro kvalitní logování. Informace je třeba zobrazovat v dostatečné míře podrobnosti, aby bylo možné najít chybu, ale v takovém objemu, aby logovací soubory zbytečně nenarůstaly. V našem případě je využita knihovna `log4net`.

`Log4net` umožňuje logování do jednoho nebo více souborů včetně jejich rotace, výstup do konzole, databáze atd. Každé třídě může být přiřazen objekt třídy `Logger`, která se poté zobrazí v logu. Součástí logování je i úroveň důležitosti hlášky, což poskytuje dobrý nástroj pro filtrování.

Konfigurace knihovny je uložena v XML souboru. Typicky se nastavuje způsob a formát výstupu a úroveň důležitost zaznamenaných informací. Pomocná třída pro snadnější použití logu je součástí přílohy F.1.

```
class Program
{
    protected static readonly ILog log = LogManager.GetLogger(typeof(Program));
    static void Main(string[ ] args)
    {
        log4net.Config.XmlConfigurator.Configure();
        log.Info("App is running");
    }
}
```

Výpis 4: Příklad použití knihovny log4net

7 Testování aplikace

Testy se navrhují a následně implementují v podobě testovacích úloh, které jednoznačně definují co se má ověřit. Testovacích procedury specifikují, jak se má test provádět, nebo se vytváří spustitelné testovací komponenty umožňující automatizaci procesu ověřování.

Výsledky provedených testů jsou systematicky zpracovávány a vadné části jsou opakovaně testovány a případně zaslány zpět do fáze analýza, návrh nebo implementace s cílem nedostatky odstranit.

7.1 Unit testy

Pod pojem unit testing se zahrnují nástroje, metodika a činnost, jejímž cílem je ověřování správné funkčnosti dílčích částí neboli jednotek zdrojového textu. Za jednotku se považuje samostatně testovatelná část aplikačního programu. Z pohledu objektově orientovaného programování je jednotkou obvykle třída.

Vývojové prostředí Microsoft Visual Studio 2008 poskytuje vestavěné nástroje pro unit testy. S jejich pomocí lze vytvořit sadu automatizovaných procedur pro ověření funkčnosti aplikace. Tím lze kontrolovat, zda zásah do programu nezpůsobí chyby ve funkčním kódu.

Přidání testu spočívá ve vytvoření nové třídy, která bude obsahovat testovací metody. Metody se mohou zaměřovat na libovolnou oblast funkcí. Ověření určitého scénáře probíhá tak, že porovnáme testovací objekty a jejich očekávané hodnoty. Pokud se shodují, test byl úspěšný, v opačném případě je nahlášena chyba.

Finální aplikace obsahuje testy pro ověření funkce perzistence objektů. Jejich použití odstraňuje chyby způsobené špatným návrhem metod pro manipulaci s daty.

8 Instalace a provoz systému

Tato kapitola se stručně zabývá způsobem instalace a požadavky na software, který je nezbytný pro běh aplikace. Podrobnější popis se nachází v uživatelské příručce v příloze B.

8.1 Požadavky na programové vybavení

Pro spuštění aplikace je třeba zajistit následující prerekvizity:

- operační systém Microsoft Windows XP a novější nebo Windows Server 2003 a novější,
- .NET framework 3.5,
- SQL Server 2000 a novější.

8.2 Průběh instalace

Instalace systému se skládá z několika kroků:

1. spuštění instalátoru zajišťující kroky k registraci služby,
2. vytvoření databáze SQL Server,
3. importování připraveného *.sql souboru, který provede vytvoření struktury databáze a testovacích dat,
4. konfigurace uživatelských parametrů.

8.3 Provoz aplikace

Běh aplikace lze řídit ovládacími prvky seznamu služeb v příslušném dialogu operačního systému. Výsledky se ukládají do databáze, informace o aktuální činnosti jsou předmětem logovacího souboru. Odinstalace probíhá standardním způsobem.

9 Zhodnocení výsledků

Ověření přínosu řídícího systému probíhalo na základě konfigurace distribučního skladu společnosti PARAMO, a.s. v Pardubicích. Základním parametrem ovlivňujícím výdej je počet a rozmístění stop a dostupnost produktů. Tabulka 2 tyto informace zobrazuje.

Obsluha odbavuje autocisterny se specifickou kapacitou. Požadovaný objem je přímo úměrný času plnění. Pro účely simulace byly zvoleny čtyři základní kategorie vozidel, jak ukazuje tabulka 3.

V tuto chvíli jsou známy všechny důležité atributy pro popis funkce skladu. Následuje porovnání dat získaných ze skutečného provozu v rozmezí tří měsíců a údajů, které jsou výstupem aplikace. Ukázalo se, že v současném řešení lze stále nalézt rezervy pro zdokonalení pracovního procesu. V situacích, kdy se v krátké době prudce zvýší počet požadavků, dochází k nerovnoměrnému zatížení. Dostupné zdroje nejsou využity efektivně.

Implementovaný algoritmus tento dojem zlepšuje. Pohled na grafy trendů času stráveného na výdejních lávkách ukazuje utlumení mezních situací (příloha E). Přínos v podobě větší plynulosti je nezanedbatelný, pro zákazníka znamená příslib rychlejšího odbavení.

Důvody pro nasazení je však třeba dobře zvážit. Uplatnění se předpokládá pro sklady s vyšším zatížením, kde je potřeba okamžitě reagovat na změnu provozních podmínek. Mezi tyto události patří poruchové stavy, které je nutné zohlednit. Řídící systém tyto vlastnosti nabízí. Jeho použití může vést k lepší kvalitě technologického procesu.

Do budoucna se nabízí rozšíření o následující funkce:

- interaktivní konfigurace uživatelských parametrů,
- adaptivní chování – poskytnutí více přístupů k problému a volba nejvhodnějšího z nich,
- grafické znázornění průběhu a výsledku činnosti.

Číslo stopy	Maximální průtok [l/m]	Název produktu	Třída produktu
1.	1000	TOL	Topný olej
2.	1000	MN	Nafta
		TOEL	Topný olej
3.	1000	MN	Nafta
		SMN30	Nafta
4.	1000	BA95 – LÍH	Benzín
5.	1000	MN	Nafta
		TOEL	Topný olej
6.	1000	BA95 – LÍH	Benzín
7.	1000	HLBI	Benzín

Tabulka 2: Konfigurace stop

Výrobce	Model	Počet komor	Celková hmotnost [kg]	Maximální objem [l]
AVIA	A31	1	5990	2900
MAN	18.224 FL	2	18 000	12 500
ENERCO	NC32X	4	34 000	32 000
ACERBI	21L2-39	5	36 800	40 777

Tabulka 3: Kategorie vozidel

10 Závěr

Skladování a výdej pohonných látek vyžaduje sofistikovaný technologický postup. Řízení podléhá nárokům na bezpečnost a okamžitý přístup k datům. Implementovaný systém se snaží tyto podmínky dodržet. Byl prokázán pozitivní vliv na provádění výdeje a zváženy argumenty pro distribuci. Aplikaci lze také použít jako nástroj pro simulaci různých variant provozu.

Přínosem této práce bylo seznámení s technickými prostředky v oblasti řízení a automatizace a ověření teoretických znalostí získaných v průběhu studia. Praktické zkušenosti s problematikou softwarového návrhu jsou důležitým poznatkem. Aplikace zavedené metodiky usnadňuje vývoj projektu a odhaluje možné problémy.

Systém má potenciál pro budoucí rozvoj. Rozšíření v podobě vizualizace se může stát předmětem dalších výzkumů.

11 Literatura

[1] ARLOW, Jim – NEUSTADT, Ila. *UML a unifikovaný proces vývoje aplikací.* 1. vyd. Brno: Computer Press, 2003. 387 s. ISBN 80-7226-947-X.

[2] ALBAHARI, Joseph – ALBAHARI, Ben. *C# 3.0 in a Nutshell.* 3. vyd. Sebastopol: O'Reilly, 2007. 819 s. ISBN 978-0-596-52757-0.

[3] ŠARMANOVÁ, Jana. *Databázové a informační systémy: Učební text.* 1. vyd. Ostrava: VŠB – TU Ostrava, 2007. 122 s. ISBN 978-80-248-1499-5.

[4] PECINOVSKÝ, Libor. *Návrhové vzory.* 1. vyd. Brno: Computer Press, 2007. 528 s. ISBN 978-80-251-1582-4.

[5] Kolektiv autorů. *Wikipedie: otevřená encyklopedie* [online]. (c) 2001–2010 [cit. 2010-05-04]. Dostupný z WWW: http//www.wikipedia.org/.

A Obsah přiloženého CD

Na přiloženém CD naleznete:

- text bakalářské práce,
- kompletní datovou analýzu,
- uživatelskou příručku,
- programátorskou příručku,
- zdrojové kódy aplikace.

B Uživatelská příručka

B.1 Instalace

Instalace se zahájí spuštěním souboru `setup.exe` z přiloženého CD. V dialogovém okně je možné specifikovat cílový adresář pro kopírování souborů.

B.2 Konfigurační soubor

Konfigurační XML soubor v instalačním adresáři obsahuje parametry pro ovlivnění simulace.

tamasConnectionString: nastavení databázového připojení.

avgVehiclesPerDay: průměrný počet vozidel za jeden den.

groundingTime: čas potřebný k uzemnění vozidla v sekundách.

loadingBayFailureProbability: pravděpodobnost poruchy stopy.

productFailureProbability: pravděpodobnost poruchy produktu.

generalFailureProbability: pravděpodobnost poruchy celého skladu.

loadingBayFailureMaxDuration: maximální doba trvání poruchy stopy v minutách.

productFailureMaxDuration: maximální doba trvání poruchy produktu.

generalFailureMaxDuration: maximální doba trvání poruchy skladu.

B.3 Kontrolní výpisy

Zde je uveden příklad logovacích záznamů o průběhu aktuální činnosti. Zobrazují se informace o naplnění každé komory a aktuální stav fronty požadavků. První číslo u jednotlivé stopy uvádí počet vozidel na terminálu, druhé zákazníky čekající u vjezdu.

```
2010-05-05 19:55:05,723 [3] INFO  2E90227 loaded, LB1: 0/0, LB2:
1/1, LB3: 1/1, LB4: 0/0, LB5: 0/0, LB6: 0/0, LB7: 1/1, total: 3
2010-05-05 19:55:05,833 [3] INFO  Loading bay: 3,
vehicle: 1A88770, chamber: 1, product: 5 (SMN30)
2010-05-05 19:55:05,991 [7] INFO  1H44114 loaded, LB1: 0/0, LB2:
1/1, LB3: 0/0, LB4: 0/0, LB5: 0/0, LB6: 0/0, LB7: 1/1, total: 2
2010-05-05 19:55:06,100 [7] INFO  Loading bay: 7,
vehicle: 2E34126, chamber: 2, product: 1 (HLBI)
2010-05-05 19:55:06,158 [2] INFO  1E03333 loaded, LB1: 0/0, LB2:
1/1, LB3: 0/0, LB4: 0/0, LB5: 0/0, LB6: 0/0, LB7: 0/0, total: 1
```

C Programátorská příručka

Účelem tohoto manuálu je seznámení s aplikací z pohledu programátora. Uvádí strukturu projektu a význam jednotlivých částí. Součástí je i příklad nastavení informačních výpisů pro záznam do konzole a souboru.

C.1 Adresářová struktura

```
\                               - design a instalátor služby
    \lib                        - knihovna pro logování
    \Model                      - doménové objekty
    \Properties                 - konfigurační atributy
    \Repository                 - implementace datové vrstvy
    \Utility                    - pomocné třídy
        \Data                   - správa připojení k databázi
        \LoggingService         - logovací služba
        \Threading              - synchronizované kolekce
```

C.2 Konfigurace služby pro logování

```
<configSections>
  <section name="log4net" type="log4net.Config.Log4NetConfigurationSectionHandler, log4net" />
</configSections>
<log4net>
  <appender name="LogFileAppender" type="log4net.Appender.FileAppender">
    <param name="File" value="log\log.txt" />
    <param name="AppendToFile" value="true" />
    <layout type="log4net.Layout.PatternLayout">
      <param name="Header" value="" />
      <param name="Footer" value="" />
      <param name="ConversionPattern" value="%d [%t] %-5p %m%n" />
    </layout>
  </appender>
  <appender name="ConsoleAppender" type="log4net.Appender.ConsoleAppender" >
    <layout type="log4net.Layout.PatternLayout">
      <param name="Header" value="[Header]\r\n" />
      <param name="Footer" value="[Footer]\r\n" />
      <param name="ConversionPattern" value="%d [%t] %-5p %m%n" />
    </layout>
  </appender>
  <root>
    <level value="DEBUG" />
    <appender-ref ref="LogFileAppender" />
    <appender-ref ref="ConsoleAppender" />
  </root>
</log4net>
```

Výpis 5: Konfigurace knihovny log4net

D Datový slovník

Atribut	Datový typ	Klíč	NULL	Index	Popis
fuelStoreId	int	A	N	A	číslo skladu
fuelStoreName	nvarchar(255)	N	N	N	název skladu
address1	nvarchar(100)	N	A	N	ulice
address2	nvarchar(100)	N	A	N	město
address3	nvarchar(100)	N	A	N	PSČ
maxLitersPerMinute	decimal(12, 3)	N	A	N	omezení průtoku
cancelled	bit	N	A	N	příznak aktivity

Tabulka 4: Tabulka FuelStore

Atribut	Datový typ	Klíč	NULL	Index	Popis
loadingBayId	int	A	N	A	číslo stopy
maxLitersPerMinute	decimal(12, 3)	N	A	N	maximální průtok
capacity	int	N	A	N	kapacita fronty
fuelStoreId	int	N	A	A	cizí klíč – tab. FuelStore

Tabulka 5: Tabulka LoadingBay

Atribut	Datový typ	Klíč	NULL	Index	Popis
loadingBayId	int	A	N	A	cizí klíč – tab. LoadingBay
productId	int	A	N	A	cizí klíč – tab. Product

Tabulka 6: Tabulka ProductOnLoadingBay

Atribut	Datový typ	Klíč	NULL	Index	Popis
productId	int	A	N	A	číslo produktu
productName	nvarchar(50)	N	N	N	název produktu
localName	nvarchar(50)	N	A	N	místní název
shortName	nvarchar(6)	N	A	N	zkratka
altName	nvarchar(50)	N	A	N	alternativní název
avgDensity	decimal(6, 3)	N	A	N	průměrná hustota
productClassId	int	N	A	A	cizí klíč – tab. ProductClass

Tabulka 7: Tabulka Product

Atribut	Datový typ	Klíč	NULL	Index	Popis
productClassId	nvarchar(10)	A	N	A	název třídy produktu
description	nvarchar(50)	N	A	N	popis

Tabulka 8: Tabulka ProductClass

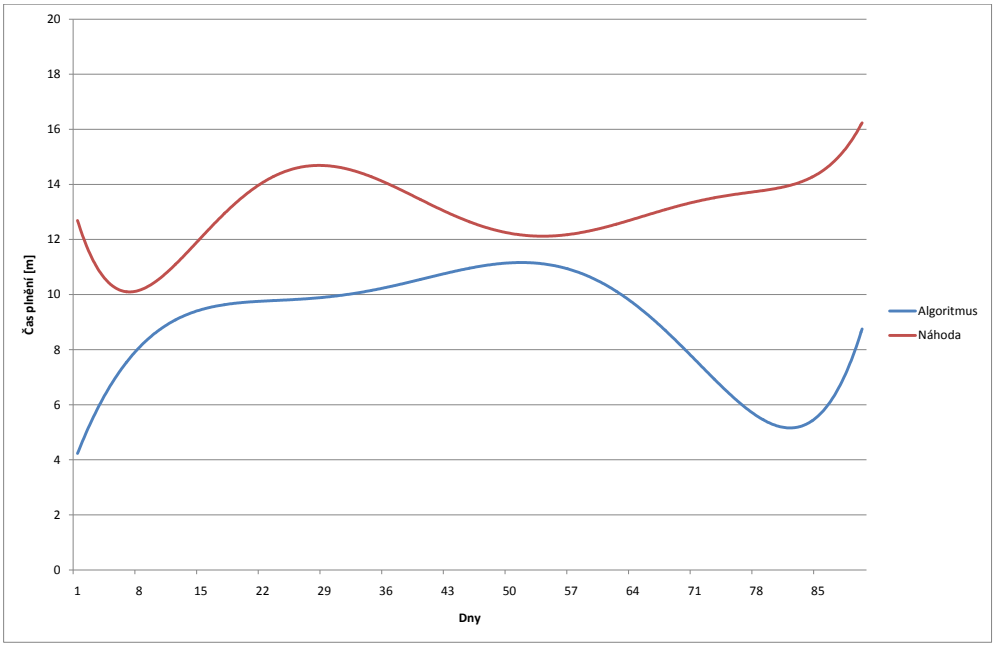
Atribut	Datový typ	Klíč	NULL	Index	Popis
vehicleId	int	A	N	A	číslo vozidla
regNumber	nvarchar(20)	N	N	N	registrační značka
vehicleType	nvarchar(10)	N	A	N	typ vozidla
manufacturer	nvarchar(30)	N	A	N	výrobce
model	nvarchar(30)	N	A	N	model
numChambers	int	N	A	N	počet komor
numAxles	int	N	A	N	počet náprav
limitWeight	decimal(12 ,3)	N	A	N	užitečná hmotnost
totalWeight	decimal(12 ,3)	N	A	N	celková hmotnost
emptyWeight	decimal(12, 3)	N	A	N	pohotovostní hmotnost
maxVolume	decimal(12, 3)	N	A	N	celkový objem
currentState	nvarchar(15)	N	A	N	umístění na terminálu
lastLoading	DateTime	N	A	N	čas posledního plnění

Tabulka 9: Tabulka Vehicle

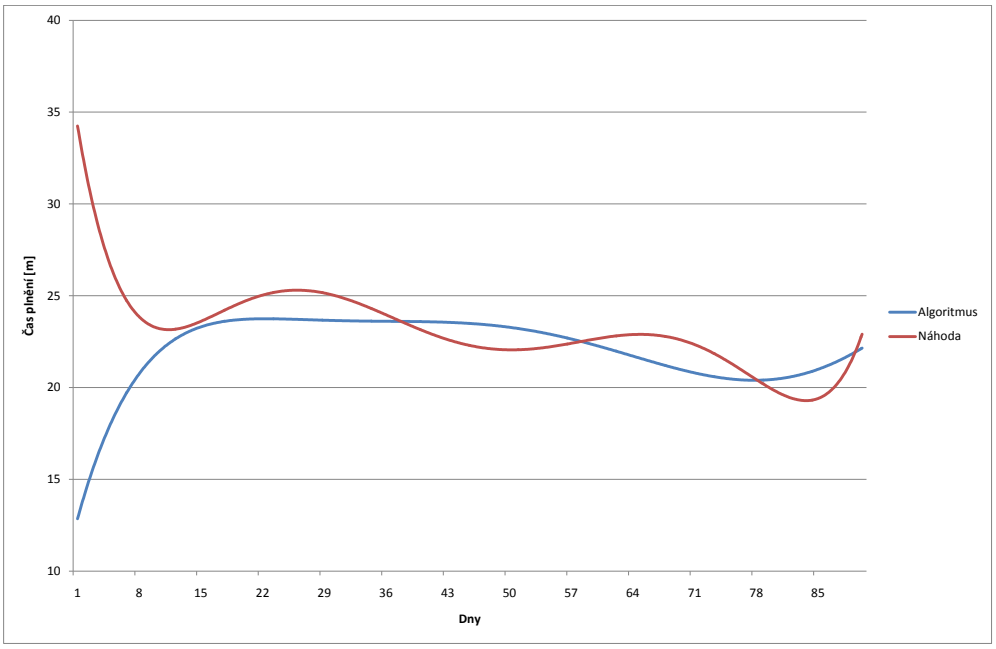
Atribut	Datový typ	Klíč	NULL	Index	Popis
chamberId	int	A	N	A	číslo komory
volume	decimal(12, 3)	N	A	N	objem
vehicleId	int	A	N	A	cizí klíč – tab. Vehicle

Tabulka 10: Tabulka VehicleChamber

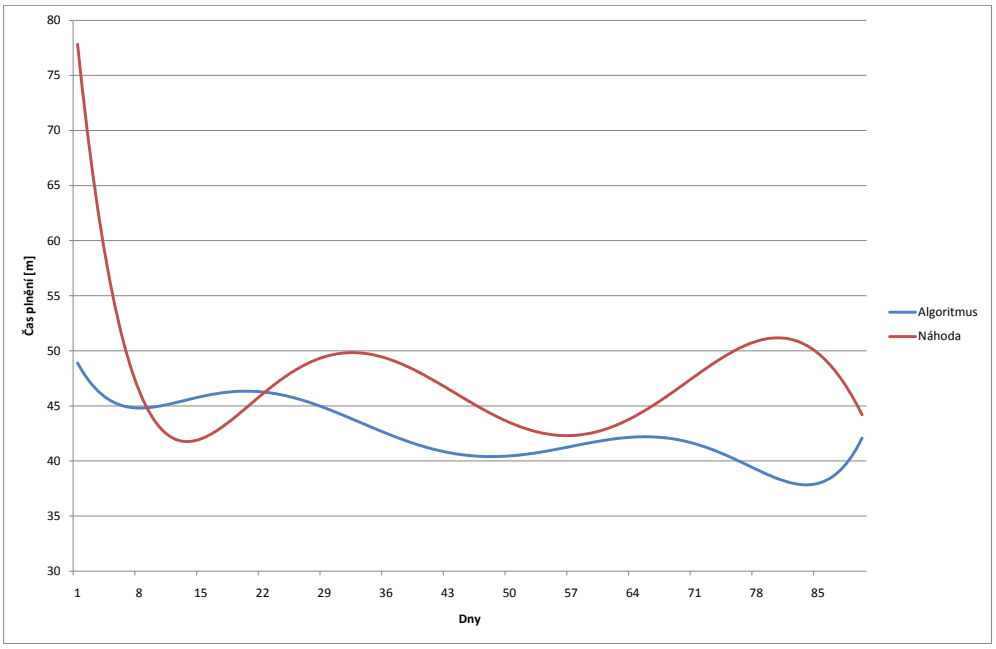
E Grafy průběhu plnění



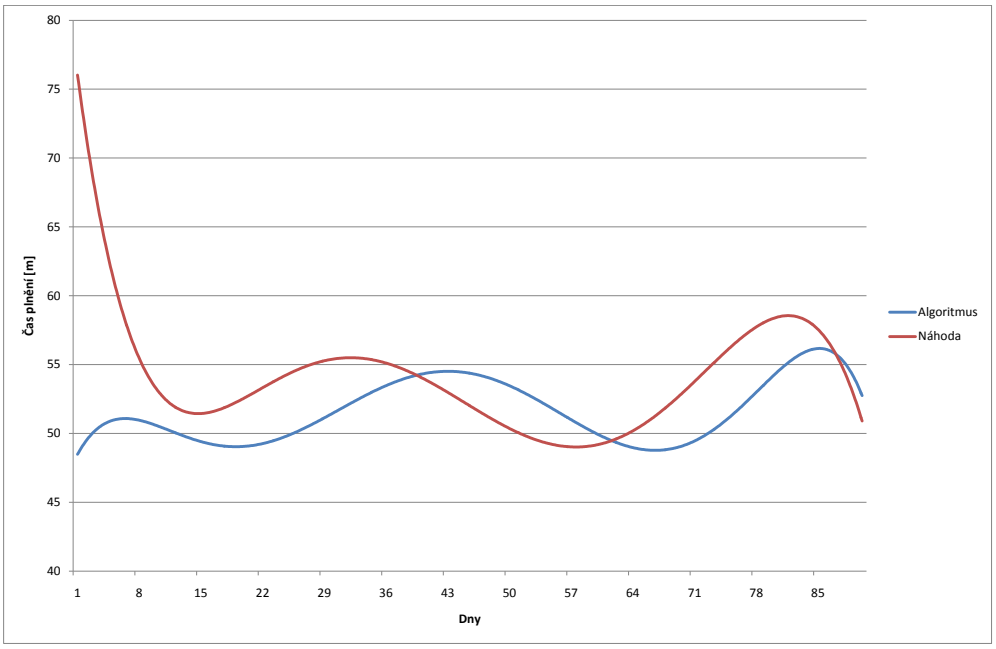
Obrázek 12: Časy plnění autocisteren o objemu 2900 l



Obrázek 13: Časy plnění autocisteren o objemu 12 500 l



Obrázek 14: Časy plnění autocisteren o objemu 32 000 l



Obrázek 15: Časy plnění autocisteren o objemu 40 777 l

F Zdrojové kódy

F.1 Třída Logger

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using log4net;
using log4net.Config;

namespace Utility
{
    namespace LoggingService
    {
        public static class Logger
        {
            private static readonly ILog logger = LogManager.GetLogger(typeof(Logger));

            static Logger()
            {
                XmlConfigurator.Configure();
            }

            public static void LogFormat(LogLevel level, String log, params object[] args)
            {
                Log(level, string.Format(log, args));
            }

            public static void Log(LogLevel level, String log)
            {
                switch (level)
                {
                    case LogLevel.INFO:
                        if (logger.IsInfoEnabled) logger.Info(log);
                        break;

                    case LogLevel.WARN:
                        if (logger.IsWarnEnabled) logger.Warn(log);
                        break;

                    case LogLevel.ERROR:
                        if (logger.IsErrorEnabled) logger.Error(log);
                        break;

                    case LogLevel.FATAL:
                        if (logger.IsFatalEnabled) logger.Fatal(log);
                        break;

                    case LogLevel.DEBUG:
                        if (logger.IsDebugEnabled) logger.Debug(log);
                        break;
                }
```

```
            }
        }
    }
}
```

Výpis 6: Třída Logger

F.2 Třída Repository

```
using System;
using System.Collections.Generic;
using System.Data.Linq;
using System.Linq;
using System.Linq.Expressions;
using System.Text;

namespace Repository
{
    public class Repository<T> : IRepository<T> where T : class, IDbTable
    {
        readonly TamasDataContext dataContext;

        public Repository()
        {
            dataContext = new TamasDataContext();
        }

        public virtual  T GetById(int id)
        {
            var itemParameter = Expression.Parameter(typeof(T), "item");

            var whereExpression = Expression.Lambda<Func<T, bool>>
                (
                Expression.Equal(
                    Expression.Property(
                        itemParameter,
                        typeof(T).GetPrimaryKey().Name
                        ),
                    Expression.Constant(id)
                    ),
                new[] { itemParameter }
                );

            var item = GetAll().Where(whereExpression).SingleOrDefault();

            if (item == null)
            {
                throw new PrimaryKeyNotFoundException(string.Format("No {0} with primary
                    key {1} found",
                                                                    typeof(T).FullName, id));
            }
```

```
    return item;
}

public virtual IQueryable<T> GetAll()
{
    return dataContext.GetTable<T>();
}

public virtual void InsertOnSubmit(T entity)
{
    GetTable().InsertOnSubmit(entity);
}

public virtual void DeleteOnSubmit(T entity, bool logicalDelete)
{
    if (logicalDelete)
    {
        entity.ModifiedDate = DateTime.Now;
    }
    else
        GetTable().DeleteOnSubmit(entity);
}
public virtual void DeleteOnSubmit(T entity)
{
    DeleteOnSubmit(entity, true);
}

public virtual void DeleteAllOnSubmit(IEnumerable<T> entities, bool logicalDelete)
{
    if (logicalDelete)
    {
        DateTime current = DateTime.Now;
        foreach (var entity in entities)
        {
            entity.ModifiedDate = current;
        }
    }
    else
        GetTable().DeleteAllOnSubmit(entities);
}
public virtual void DeleteAllOnSubmit(IEnumerable<T> entities)
{
    DeleteAllOnSubmit(entities, true);
}

public virtual void SubmitChanges()
{
    dataContext.SubmitChanges();
}

public virtual ITable GetTable()
{
    return dataContext.GetTable<T>();
}
```

```
    }
}
```

Výpis 7: Třída Repository

F.3 Třída BlockingPriorityQueue

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;

namespace Utility.Threading
{
    public class BlockingPriorityQueue<P, V>
    {
        private SortedDictionary<P, Queue<V>> list = new SortedDictionary<P, Queue<V>>();

        public void Enqueue(P priority, V value)
        {
            lock ( list )
            {
                Queue<V> q;
                if (! list .TryGetValue(priority , out q))
                {
                    q = new Queue<V>();
                    list .Add( priority , q);
                }
                q.Enqueue(value);
                Monitor.Pulse( list );
            }
        }

        public V Dequeue()
        {
            KeyValuePair<P, Queue<V>> pair;
            lock ( list )
            {
                while ( list .Count() == 0)
                {
                    Monitor.Wait( list );
                }
                pair = list . First ();
                V v = pair.Value.Dequeue();
                if (pair.Value.Count == 0)
                {
                    list .Remove(pair.Key);
                }
                return v;
            }
```

```
        }

        public int Count()
        {
            int count = 0;
            lock ( list )
            {
                foreach (KeyValuePair<P, Queue<V>> pair in list)
                {
                    count += pair.Value.Count();
                }
                return count;
            }
        }
    }
}
```

Výpis 8: Třída BlockingPriorityQueue

F.4 Třída Worker

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;

namespace Utility.Threading
{
    public abstract class Worker
    {
        public static TimeSpan DefaultInterval = TimeSpan.FromSeconds(5);

        protected volatile bool serviceStarted = false;
        protected TimeSpan interval = DefaultInterval;
        private ManualResetEvent pauseEvent = new ManualResetEvent(true);
        private Thread thread;

        public Worker() { }

        public Worker(TimeSpan interval)
        {
            this.interval = interval;
        }

        public void Start()
        {
            thread = new Thread(ExecuteTask);
            serviceStarted = true;
            thread.Start();
        }
```

```
        public void Stop()
        {
            serviceStarted = false;
            thread.Join() ;
        }

        public void Pause()
        {
            pauseEvent.Reset();
        }

        public void Resume()
        {
            pauseEvent.Set();
        }

        private void ExecuteTask()
        {
            while (serviceStarted)
            {
                pauseEvent.WaitOne(Timeout.Infinite);
                Work();
                if (serviceStarted)
                {
                    Thread.Sleep(interval);
                }
            }
            Thread.CurrentThread.Abort();
        }

        public abstract void Work();
    }
}
```

Výpis 9: Třída Worker